ESD-TR-68-454

(Final Report)

ESLFE

JOVIAL APPLICATION QUESTIONNAIRE

William M. O'Brien

December 1968

COMMAND SYSTEMS DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

(Prepared under Contract No. F19628-68-C-0301 by Data Dynamics, Inc. 9800 S. Sepulveda Boulevard, Los Angeles, California 90045.)

LEGAL NOTICE

When U. S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy.

ESD-TR-68-454

(Final Report)

JOVIAL APPLICATION QUESTIONNAIRE

William M. O'Brien

December 1968

COMMAND SYSTEMS DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts

(Prepared under Contract No. F19628-68-C-0301 by Data Dynamics, Inc. 9800 S. Sepulveda Boulevard, Los Angeles, California 90045.)

FOREWORD

The questionnaire presented herein was the primary tool used to evaluate JOVIAL (J3) as specified in AFM 100-24. The use of this manual is limited in scope due to its primary purpose, that of providing numerical data for the evaluation of the language. The textual material was presented in great detail because it was felt that the reader would not necessarily be familiar with the J3 dialect of JOVIAL.

The questionnaire was produced primarily by William O'Brien of Data Dynamics as part of Project 6917, Task 04, under contract F19628-68-C-0301 for Electronic Systems Division, Air Force Systems Command. The project monitor was Capt. Martin J. Richter.

This questionnaire has been reviewed and approved by Hq. USAF and is assigned the reports control symbol HAF-D-86-(OT).

Martin J. Richter

MARTIN J. RICHTER, Capt., USAF
Project Monitor

William F. Heisler

WILLIAM F. HEISLER, Col., USAF
Chief, Command Systems Division
Directorate of Planning & Technology

ABSTRACT

The JOVIAL Application Questionnaire was produced as a vehical to gather information regarding JOVIAL users experience with the language and the environment in which JOVIAL was being used. This information is to be utilized to evaluate JOVIAL (J3) computer programming languages as specified in AFM 100-24. The questionnaire contains: Instruction on how to fill out the questionnaire; general questions about the application being programmed in JOVIAL; the hardware and operating systems being used; background information; specific questions about each JOVIAL feature with regard to the conformance of the specification of the feature to AFM 100-24 and the extent of utilization of the feature. In addition, the questionnaire contains a detailed description of each JOVIAL feature as specified in AFM 100-24 as a convenient reference to the users of a different JOVIAL language dialect.

TABLE OF CONTENTS

Table of Contents (Continued)

SECTION I

INTRODUCTION

On 15 June 1967, Air Force Manual 100-24 was published. This manual, entitled "Standard Computer Programming Language for Air Force Command and Control Systems", defines a JOVIAL (J3) dialect and establishes that dialect as the standard language to be used in USAF command and control applications. Throughout this document, this dialect will be alternately referred to as "J3" or Standard J3.

The primary purposes of this JOVIAL Application Questionnaire (JAQ) are

- to determine the deviations of the JOVIAL dialects currently being used in command and control programming applications from Standard J3, and

- to determine usage rates of those feature in these JOVIAL dialects which do not deviate significantly from Standard J3.

By "features" we mean those J3 language capabilities which enable the expression of solutions to computer-oriented problems. Examples of J3 features are in the Item Declaration, the FOR statement, and the OPEN, SHUT, INPUT, OUTPUT file manipulation statements.

Secondarily, the JAQ solicits information regarding specific functional characteristics associated with your application, operating millieu, programming experience and the like.

This questionnaire is organized into five sections:

Project Identification	(Section 3.1)
Hardware Configuration	(Section 3.2)
Operating System	(Section 3.3)
Background Information	(Section 3.4)
JOVIAL Application Information	(Section 3.5)

SECTION II

INSTRUCTIONS

Sections 3.1 through 3.4 of the JAQ are self-explanatory.

Section 3.5 is organized around Standard J3 features. Text is incorporated to describe each feature both in syntactic terms -- the grammatical structure of the J3 "sentences" by which the feature is called out in J3 programs -- and in semantic terms -- the effect produced on a program's enviornment by the use of the feature. We would greatly appreciate your reading this text carefully so that you may accurately compare the syntax and semantics specifications of Standard features against the syntax and semantics of the features in your version of JOVIAL.

In most instances, questions of the following form accompany the text for a given feature:

1. In your version of JOVIAL, is the "such-and-such" feature

Qa
included exactly as described above? _____

Qb
included, but differing in one or more particulars from the Standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the Standard:

_____

_____

_____

Qc

not included at all? If not included, kindly state the reason or reasons why: _____

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Qd

How many times is the "such-and-such" feature used in your programs?

By count _____

By estimate _____

Questions Qa, Qb, Qc are answered simply with YES or NO in the spaces provided; clearly, two of the three should be answered NO and the third YES. If question Qb is answered YES, space is provided for you to tell us how this feature is implemented in your version of JOVIAL. Also, we solicit information on why your implementation differs from the standard. Examples of valid responses here are "unknown" -- it was just done that way" and "the feature as specified in the standard has no meaning with respect to our equipment". If question Qc is answered YES, we ask again for reasons why, and also we ask if there are other mechanisms in your system, such as subroutines, say, which perform the functions of the feature. Please understand that deviations of your version of JOVIAL from the standard in no way reflect on your competency; there are no right or wrong answers and, in fact, there may be exceedingly good reasons why such deviations exist.

If the space provided for the purpose of describing these deviations is insufficient, please attach your descriptions to the page of the Questionnaire where the question occurs.

The following hypothetical deviations and possible responses to questions Qa, Qb, Qc are shown by way of example:

Example A

Section 3.5.1.1.1 describes the octal constant in J3. Your implementation concurs in every particular. You will then answer

Qa	YES
Qb	NO
Qc	NO

and proceed to answer Qd.

Example B

When status items are declared via item declarations, the standard permits the programmer to specify optionally the number of bits required to register the status item. Your version does not permit this. Your response might read:

In your version of JOVIAL, is the status item feature

Qa
included exactly as described above? NO

Qb
included, but differing in one or more particulars from the standard described above? YES

If yes, please describe your implementation of the feature:

Our version of JOVIAL is identical with the standard's except that the optional size specification is not permitted.

Also, please state the reason or reasons why your implementation differs from the standard?

It was felt that this mechanism would serve little pupose in our application.

Qc
not included at all? If not included, kindly state the reason or reasons why:

NO

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Example C

The file manipulation features described in Section 3.5.3.3 have not been implemented in your version at all.

In your version of JOVIAL, are the file manipulation features (OPEN, SHUT, INPUT, OUTPUT, POS)

Qa
included exactly as described above?

NO

Qb
included, but differing in one or more particulars from the standard described above?

NO

If yes, please describe your implementation of the feature.

Also, please state the reason or reasons why your implementation differs from the standard?

Qc not included at all? If not included, kindly state the reason or reasons why: YES

It was felt that these mechanisms were inadequate for our applications for the following reasons " . . ."

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

We use subroutines to perform the I/O tasks. These routines have the following functional makeup: ". . . "

Question Qd may be answered either by count or by estimate as indicated. If question Qc is answered YES, that is if your implementation of a feature differs somewhat from the standard, answer question Qd on the basis of the number of occurrences of the feature as implemented in your version. If question Qc is answered YES -- your version of JOVIAL does not include this feature at all -- do not answer question Qd.

Section 3.5.7 is included so that you may describe features in your version of JOVIAL which are not mentioned in the standard. (Perhaps you have a MERGE "verb" or a data editing statement to convert binary data to and from Hollerith, to give two examples).

SECTION III

QUESTIONNAIRE

3.1 APPLICATION IDENTIFICATION

This section is included in the questionnaire in order to provide identification for the particular application, or project, under investigation and to describe some of its pertinent characteristics.

Please provide the following requested information:

Application name

Application description (short prose description of purpose of the application)

Total number of JOVIAL statements in the collection of source programs in this application _____

Physical program characteristics

Number of on-line magnetic tapes used _____
Number of on-line discs used _____
Number of on-line card readers _____
Number of on-line card punches _____
Number of on-line printers _____

Operating characteristics

Under which of the three operating modes has this application been run. Circle appropriately.

Batch processing	YES	NO
Time Sharing	YES	NO
Real Time	YES	NO

3.2 HARDWARE CONFIGURATION

This section is included in the questionnaire in order to provide a description of the hardware configuration under which this application is generally executed. If more than one device of the same type is present on the hardware configuration provide this information in those portions of the questionnaire indicated. If still more space is required, use the space provided at the end of this section.

Computer manufacturer and model _____

Storage

- High speed memory _____ words
  - Size _____ six bit bytes
  - Word size if applicable _____ bits

- Input/Output devices _____
  - Drum
    - Manufacturer and model _____
    - Storage capacity _____ six bit bytes
      _____ bits
    - Quantity on this configuration _____
  - Disc
    - Manufacturer and model _____
    - Storage capacity _____ six bit bytes
      _____ bits
    - Quantity on this configuration _____
  - Magnetic tapes
    - Manufacturer and model _____
    - Recording densities permitted _____ bits/in
      _____
    - Quantity on this configuration _____
    - Manufacturer and model _____
    - Recording densities permitted _____ bits/in
      _____
    - Quantity on this configuration _____
  - Card punches
    - Manufacturer and model _____
    - Card punch speed _____ cards/min
    - Quantity on this configuration of this type _____

Manufacturer and model _____
Card punch speed _____ cards/min

Quantity on this configuration of this type _____

Card reader
Manufacturer and model _____
Card read speed _____ cards/min
Quantity on this configuration _____

Printer
Manufacturer and model _____
Printer speed _____ lines/min

Quantity on this configuration _____

Printer
Manufacturer and model _____
Printer speed _____ lines/min

Quantity on this configuration _____

Paper tape reader
Manufacturer and model _____
Tape read speed _____ frames/sec
Quantity on this configuration _____

Paper tape reader
Manufacturer and model _____
Tape read speed _____ frames/sec
Quantity on this configuration _____

Paper tape punch
Manufacturer and model _____
Tape punch speed _____ frames/sec
Quantity on this configuration _____

Paper tape punch
Manufacturer and model _____
Tape punch speed _____ frames/sec
Quantity on this configuration _____

The following space is provided for pertinent portions of the hardware configuration that has not been specifically acquired or solicited in the current section of the questionnaire.

3.3 OPERATING SYSTEM

This section is included in the questionnaire to provide certain characteristics of the operating system under which this application is executed.

Name of operating system _____

Name of developer _____

Mode of operating system _____

- Batch processing _____
- Time sharing _____
- Real time _____

Under what operating system mode is the current application executed. _____

3.4 BACKGROUND

This portion of the questionnaire will be used to collect ancillary application information. This section will concern itself with management attitudes toward JOVIAL, programmer experience levels and any other attitude that effects the use of the JOVIAL language.

1. Why was JOVIAL used to program this application

   a. To remain source language compatible with other related project
      YES NO

   b. Because it was the official Air Force language
      YES NO

   c. Because it was the best language suited for the project
      YES NO

If for any other reason give complete explanation.

2. What was the average experience (in years) of the programmers on this this project?

3. What was the average JOVIAL experience (in years) of the programmers on this project?

4. What JOVIAL dialect is used in your applications (J1, J2, J3, J4. . .)?

5. What is the compilation speed in statements per minute?

6. Is the compiler generated code relocatable?

3.5 APPLICATION QUESTIONNAIRE

3.5.1 Data Representation

All data in Standard J3 have associated with them certain J3-specific attributes. Each attribute can be regarded as taking values which are J3 data properties. For example, the J3 data properties "item", "array", "table", "string", and "file", all of which are related in the sense that they deal with data structure -- may be thought of as the set of values taken by the J3 structure attribute. The process of specifying data representation in J3, though not usually described in this way, can be conveniently expressed in terms of the assignment of data property values to data attributes. This section is organized on the basis of the attributes and attribute values which underly data representation in Standard J3. Information is solicited on your implementation and usage rates of J3 attributes and their values, and also on the mechanisms (e.g., Item Declaration) by which values are assigned to attributes.

3.5.1.1 Structure Attribute = "Item"

Item data are structurally the simplest kind of data in Standard J3. Associated with all items is the item type attribute. For a particular item, the item type attribute may take one of the following values:

"Integer"
"Fixed Point"
"Floating Point"
"Hollerith"
"Standard Transmission Code"
"Boolean"
"Status"
"Dual"

Depending on the value of the item type attribute, other attributes may also be associated with item data. These are:

Size
Scale
Sign
Range
Round

Questions relating to item type attribute values and to attributes associated with item data are asked in paragraphs 3.5.1.1.4 through 3.5.1.1.15. The value

"item" is assigned the structure attribute of a datum

- by the use of one of the J3 constant forms,
- by the means of the Item Declaration, and
- by the operation of certain functional modifiers on data.

(Functional modifiers are described in Section II.) Every J3 constant but one may be assigned an item type on the basis of its grammatical representation as a string of J3 signs and independent of its context in a J3 statement. Values are assigned to the size, scale, and sign attributes by this means also (except for status and Boolean constants). For example, the constant -31.5A1 is assigned item type = "Fixed Point", sign = "signed", size = 7 and scale = 1 simply by its appearance. The one constant which cannot be assigned an item type out of context is

3.5.1.1.1 The Octal Constant

Octal constants have the form:

O (octal'digits) (e.g., O (30715))

where octal'digits denotes a string of one or more of the signs 0 1 2 3 4 5 6 7 . Octal constants are used to establish values for item data in terms of bit patterns.

In your version of JOVIAL, is the octal constant feature

Q1 included exactly as described above? _____

Q2 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

Q3 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q4 how many times are octal constants used in your programs?

By count _____

By estimate _____

The next two paragraphs refer to the item declaration and the mode directive as mechanisms for assigning attribute values.

3.5.1.1.2 Item Declaration

Standard J3 permits two forms for the item declaration

3.5.1.1.2.1 Form 1:

ITEM identifier (e.g., ITEM AA

basic'item'description	I 48 S
[round'specification]	R
[range'specification]	25...8000
[item'preset'specification]$	P 4095)

(Note: Square brackets [ ] are used to imply that whatever is enclosed in them is optional.)

The effect of Form 1 is

- to associate the name "identifier" with the datum declared
- to assign "item" as the value of the datum's structure attribute
- to assign a value to the item type attribute of the datum (and also to give values to the size, sign, and scale attribute where applicable) on the basis of information making up the basic'item'description, and
- to establish values for the range, round, preset attributes from their respective specifications (if present).

3.5.1.1.2.2 Form 2:

ITEM identifier [±] constant $

Form 2 is not applicable when the user wishes to specify status or Boolean as item type. The effect of form 2 is

- to associate the name "identifier" with the datum defined
- to assign "item" to the datum's structure attribute
- to assign values to the datum's item type (and where applicable, size, sign, and scale attributes) on the basis of the grammatical structure of constant.
- establish the initial value of the datum as the value represented by "constant".

In short, the "constant" of Form 2 is used to "generate" a basic'item'description and a preset'specification, e.g., ITEM AA 3H (XYZ) $ has the same meaning as ITEM AA H 3 P 3H (XYZ) $

Note that ± is only applicable if the constant is numeric.

In your version of JOVIAL, is the second form of the Item Declaration (e.g., ITEM AA 3H (XYZ) $)

Q5 included exactly as described above? _____

Q6 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard:

Q7 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q8 how many times is this feature used in your programs?

By count _____

By estimate _____

3.5.1.1.3 The Mode Directive

The mode directive provides an automatic Item Declaration facility. It is a compiler directive, active only during program compilation.

Form: MODE	basic'item'description	(e.g., MODE I 48 S
	[round'specification]	R
	[range'specification]	25...8000
	[item'preset'specification]	P 4095)

- The effect of the Mode Directive begins at the point where it occurs in the source code and ends at the next occurrence of a Mode Directive or the end of the source code (TERM card), if there is no subsequent Mode Directive.
- All identifiers encountered within the scope of effect of the Mode Directive, which are declared no where else in the program, are declared by the description and optional specifications following MODE - the compiler uses the Mode Directive to "construct" Item Declarations for identifiers not defined by Item Declarations.
- A predefined Mode Directive is invoked by the compiler immediately prior to starting the compilation. It has the form

  MODE I δ S $

  where δ is the (implementation dependent) number of bits in the computer "word" I means item type = "integer" and S means sign attribute = "signed".

In your version of JOVIAL, is the Mode Directive feature

Q9 included exactly as described above? _____

Q10 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard:

Q11 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q12 how many item data are declared by the Mode Directive used in your programs?

By count _____

By estimate _____

Q13 how many Mode Directives are explicitly used in your programs (exclusive of the automatic Mode Directive MODE I δ S $)

By count _____

By estimate _____

Q14 how many items, no matter how declared, appear in your programs?

By count _____

By estimate _____

In subsequent paragraphs, where questions regarding item types appear, the basic'item'descriptions which specify these types in Item Declarations are exhibited; forms of associated constants are also exhibited. When encountered in basic'item' descriptions or constant forms:

- n is an unsigned integer constant (e.g., 23) which establishes a value for the size attribute
- m is an optionally signed integer constant (e.g., -6, +6, 6) which establishes a value for the scale attribute
- s is either of the letters S or U and is the value of the sign attribute
- digits denotes a string of one or more of the signs {0 1 2 3 4 5 6 7 8 9}

Also, certain letters denote values for the item type attribute:

I	denotes	"Integer"
A		"Fixed Point"
D		"Dual"
F		"Floating Point"
H		"Hollerith"
T		"Standard Transmission Code"
S		"Status"
B		"Boolean"

3.5.1.1.4 Item Type = "Integer"

basic'item'description: I n s (e.g., I 36 U)
constant form: digits (e.g., 25)
or digits E[±]digits (e.g., 25 E10)
meaning Integer data take "whole numbers" as values. They are represented as binary positional numbers.

In your version of JOVIAL, is the item type "integer" feature

Q15 included exactly as described above? _____

Q16 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard:

_____

_____

_____

_____

Q17 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the function of this feature:

_____

_____

Q18 how many times is this feature used in your programs?

By count _____

By estimate _____

3.5.1.1.5 Item Type = "Fixed Point"

- basic'item'description A n s [m] (e.g., A 25 S 5)
- constant form: floating'constant [±] digits (floating'constant stands for any one of the constant forms in paragraph I A 8)
- meaning: Fixed Point items take as values numbers with both integer and fractional parts. They are represented as binary positional numbers. If the scale value m is not included in the basic'item'description, then the Fixed Point datum is equivalent to an integer datum declared with I n s. If m is included and is positive, it denotes the number of fractional bits of the item. If m is negative, the bits comprising the item are treated as high order integer bits.

In your version of JOVIAL, is the Fixed Point item feature

Q19 included exactly as described above? _____

Q20 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard:

_____

_____

_____

_____

Q21 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q22 how many times are Fixed Point items declared in your programs?

By count _____

By estimate _____

Q23 how many times are Fixed Point items declared in your programs with either the scale excluded (e.g., A 25 S) or with scale 0 (e.g., A 25 S 0)

By count _____

By estimate _____

3.5.1.1.6 The Sign Attribute

Values for the sign attribute are given explicitly for Integer and Fixed Point data only. (The sign attribute is also associated with Floating Point data; in Standard J3 the value taken by this attribute is predefined as "S"). When a numeric item is signed (sign attribute = S), sign information, as well as magnitude information, is carried with the item. If the item is unsigned, only magnitude information is carried. The sign information is carried in a binary digit. If the binary digit is 1, the numberic item is negative; if it is 0, the numeric item is positive.

Examples:

ITEM AA I 48 S $ declares the sign attribute of AA = S (signed)

ITEM BB A 47 U 0 $ declares the sign attribute of BB = U (unsigned)

In your version of JOVIAL, is the sign specification feature of "Integer' and Floating Point numbers

Q24 included exactly as described above? _____

Q25 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard:

_____

_____

_____

_____

Q26 how many Integer and Fixed Point items are declared signed in your programs?

By count _____

By estimate _____

3.5.1.1.7 Item Type = "Dual"

- Basic item description " D I n s (e.g., D I 12 S)
  or D A n s m (e.g., D A 12 S 3)

constant form: D (integer constant, integer constant)
(e.g., D (25, 921))
or
D (fixed constant, fixed constant)
(e.g., D (26.5A1, -127.25A2))
or
D (octol constant, octal constant)
(e.g., D (O (707), O (707)))

(integer constant and fixed constant denote the constant forms shown in paragraphs I A 4 and I A 5 respectively) meaning: A dual item consists of two components. Both components must have the same item type - either both are Integer items or both are Fixed Point. Further, all other attributes associated with one component are identical to those associated with the other and these attributes have identical values for each component. Whe dual items are referenced procedurally, both components are referenced at once.

In your version of JOVIAL, is the item type "Dual" feature

Q27 included exactly as described above? _____

Q28 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q29 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the function of this feature?

_____

_____

_____

_____

Q30 how many times are "Dual" items used in your programs?

By count _____

By estimate _____

3.5.1.1.8 The Range Attribute

The range attribute is associated only with item types "integer", "Fixed Point", and "Dual". Values are explicitly provided for it when the optional 'range' specification is entered in an Item Declaration. This specification has the form

constant. . .constant (e.g., 25.5A1. . .625.75A2)

where the first and second "constant" denote respectively the minimum and maximum absolute values taken by the item. If the range'specification is not included, each "constant" may be thought of as the same value - namely, the largest positive value which can be registered by the item. Range is used in intermediate scaling.

In your version of JOVIAL, is the "Range Specification" feature

Q31 included exactly as described above? _____

Q32 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q33 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q34 how many times is the range specification used in your programs in Integer, Dual, or Fixed Point item declarations?

By count _____

By estimate _____

3.5.1.1.9 Item Type = "Floating Point"

- Basic'item'description: F
- Constant'form: digits (e.g., 314159.)
  or digits.digits (e.g., 3.14159)
  or .digits (e.g., .314159)
  or digits.E ± digits (e.g., 314159.E -5)
  or digits.digits E ± digits (e.g., 3.14159 E 0)
  or .digits E ± digits (e.g., .314159 E + 1)

- meaning: A "Floating Point" item consists of two elements. They are called the characteristic (or exrad) and the mantissa (or signicand); the characteristic may be thought of as a signed integer and the mantissa as a signed fixed point number with no integer bits. Standard J3 does not specify sizes for either the mantissa or charasteristic - these sizes are treated

as implementation dependent. The value of a Floating Point item is expressed by (2** characteristic) * mantissa.

In your version of JOVIAL, is the "Floating Point" item feature

Q35 included exactly as described above? _____

Q36 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard:

_____

_____

_____

_____

Q37 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q38 how many item data are declared as Floating Point in your programs?

By count _____

By estimate _____

3.5.1.1.10 The Round Attribute

The Round attribute is associated only with item data having item types "Integer", "Fixed Point", or "Floating Point". If the round'specification - simply the letter R - is present in an Item Declaration, the round attribute has the value "round any number which has greater precision than the declared item before it is assigned to the item". If the letter R is not present, the round attribute has the value "Truncate (i.e., do not round) any such number before assignment". Example:

If AA is declared by ITEM AA I 24 S R $, then the assignment statement AA = 64.5A1 $ produces a value of 65 for AA (i.e., 64.5A1 is "rounded up" to the integer 65".)

In your version of JOVIAL, is the "Round" feature

Q39 included exactly as described above? _____

Q40 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard:

Q41 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q42 how many times is the round specification included in Item Declarations in your program?

By count _____

By estimate _____

3.5.1.1.11 Literal Data

Data with item type "hollerith" or "Standard Transmission Code" are called literal data because they provide the facility for handling "character" data.

3.5.1.1.11.1 Item Type = "Hollerith"

- Basic item description H n (e.g., H 3)
- Constant form: digits H (string of Hollerith characters) (e.g., 3H (XYZ))
  (digits represents the number of Hollerith characters between the parentheses)
- Meaning: A "Hollerith" item takes strings of Hollerith characters as values. These characters must include at least the set of signs used in Standard J3 and may contain other implementation dependent characters.

In your version of JOVIAL, is the "Hollerith" item feature

Q43 included exactly as described above? _____

Q44 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard:

_____

_____

_____

_____

Q45 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q46 how many times are Hollerith items declared in your program?

By count _____

By estimate _____

3.5.1.11.2 Item Type = "Standard Transmission Code" (STC)

- Basic item description: T n (e.g., T 3)
- Constant form: digits T (string of STC characters) (e.g., 3T(XYZ))
  (digits represents the number of STC characters between parenthesis)
- meaning: An "STC" item takes strings of STC characters as values.

In your version of JOVIAL, is the "STC" item type feature

Q47 included exactly as described above? _____

Q48 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q49 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q50 how many times are STC items declared in your programs?

By count _____

By estimate _____

3.5.1.1.12 Item Type = "Boolean"

- Basic'item'description: B
- Constant form: 0
  or 1

• meaning: "Boolean" items register only two possible values - "true" and "false". They are represented by means of one bit. The value of the bit is 0 for "false", 1 for "true".

In your version of JOVIAL, is the "Boolean" item feature

Q51 included exactly as described above? _____

Q52 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard:

_____

_____

_____

_____

Q53 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q54 how many times are Boolean items declared in your program?

By count _____

By estimate _____

3.5.1.1.13 Item Type = "Status"

- Basic'item'description: S [m] list'of'status'constants (e.g., S 1 V(YES) V(NO))
- Constant form: V (letter) (e.g., V(A))
  or V (identifier'form) (e.g., V(A'139Z))
- meaning: The values "status" items take are shown explicitly in the "list'of'status' constant". These values are represented as unsigned integers. The first status constant in the "list'of'status'constants" is represented by 0, the second by 1, the third by 2 and so on.

In your version of JOVIAL, is the status item feature

Q55 included exactly as described above? _____

Q56 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

Q57 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q58 how many times are status items declared in your programs?

By count _____

By estimate _____

3.5.1.1.14 The Size Attribute

In the case of item types Floating Point and Boolean, values are assigned the size attribute implicitly. For all other item types, the programmer may explicitly assign a value to the size attribute.

- With respect to Integer and Fixed point items:

  If the item is unsigned, the value of size denotes the length of the magnitude of the item in binary digits.

  If the item is signed, the value of size denotes the length of the magnitude plus the binary digit used to represent the sign, i.e., length of magnitude = size - 1

- With respect to Hollerith and STC items:

  Size denotes the number of Hollerith or STC characters making up the item.

- With respect to Status items:

  Size denotes the number of binary digits making up the unsigned integer used to represent the set of associated status values.

  If m is included in the basic'item'description size is assigned m as a value. Otherwise size is determined as the smaller number of binary digits required to register the largest unsigned integer associated with any of the status constants.

With respect to "Fixed Point" and "Integer" items, in your version of JOVIAL, is the size attribute feature

Q59	included exactly as described above?	_____
Q60	included, but differing in one or more particulars from this standard described above?	_____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

Q61 not included at all? _____

If not included at all, kindly state the reason or reasons why?

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q62 how many different size values are specified for Integer amd Fixed Point items in your programs?
(Example: If size specification in all your Integer and Fixed Point Item Declarations are, say, 10, 12, and 48, and no other size values are given, answer 3.)

By count _____

By estimate _____

With respect to literal data: In your version of JOVIAL, is the "Size Attribute" feature

Q63 included exactly as described above? _____

Q64 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q65 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any which you employ to perform the functions of this feature?

Q66 How many different size values are specified for literal items in your programs?
(Example: If size specifications in all your literal Item Declarations and all your literal constants are 3, 25, and 106, say, answer 3.)

By count _____

By estimate _____

In the case of status items: In your version of JOVIAL, is the "Size Attribute" feature

Q67 included exactly as described above? _____

Q68 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q69 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform this feature?

Q70 how many times where status items declared in your programs in the size shown explicitly? (e.g., ITEM AA S 1 V(YES) V(NO) $)

By count _____

By estimate _____

3.5.1.1.15 Computer Representation of Item Data

The item data of the preceding paragraphs have been described independently of how they are to be stored in the computer memory. Thus, descriptions of data have been given in terms of "binary digits" and "characters." This section describes allocation of memory containers for item data and representation of item data within their containers.

Specification of storage allocation for item data in Standard J3 are described in terms of a Standard J3 computer memory. It is understood that the actual memory of a particular computer for which the Standard is implemented need not have the structure of the Standard J3 memory. However, the implementation should "simulate" the Standard J3 memory in the sense that a J3 program and the computer impinge on the program's environment precisely as they would if the implementation computer memory had the same structure as the Standard J3 memory.

3.5.1.1.15.1 The Standard J3 Memory

- The Standard J3 memory is composed of words, each of which has associated with it a unique address. The address is 0 or a finite positive number. Two words are said to be contiguous if their addresses differ by 1.

- Each word is a string of bits, δ in number. δ is implementation dependent. Associated with each bit in each word is a bit referencing number. Bit referencing numbers range from 0 to $\delta - 1$. Bit number 0 is called the leftmost bit in the word; bit number $\delta - 1$ is the rightmost bit.

- Superimposed over each word is a string of one or more bytes, Ψ in number. Each of the Ψ bytes in each word is a string of ω bits. Ψ and ω are implementation dependents. Bytes, like bits, are numbered from "left to right" in each word starting with byte 0 and ranging through byte $\Psi - 1$. If the number of bits in the word does not divide evenly by the number of bits per byte, the leftmost "remainder" bits in the word are bypassed, i.e., the string of bytes is right justified in the word.
  Example:
  If the number of bits in the word is 38 and if there are 6 bits/byte, byte 0 begins at bit 2 in the word.

3.5.1.1.15.2 Allocation of Item Data to the J3 Memory

Integer, Fixed-Point, Dual, Boolean, and Status items are always stored in strings of memory bits. These memory bit strings must always start and end in the same computer word. In the case of packed tables (Paragraph 3.5.1.3.6), the programmer may direct containing bit strings to begin at any point in a word. If Integer, Fixed-Point, Dual, Boolean and Status items are non-packed -- declared independently of tables, arrays, or strings -- the bit string in which they are stored is one word in length, beginning at bit 0 of the word and ending at bit $\delta - 1$ of that word.

Hollerith and STC items are allocated to strings of bytes. Byte string containers may begin in one word and end in another; in going from one word to the next, byte strings resume at byte 0 of the next word (no bytes are skipped). Non-packed Hollerith and STC items always begin in byte 0 of the first word of the byte string. If the size of a non-packed literal item is Ψ characters or less, a containing byte string of Ψ characters is allocated. If the size of the item is greater than Ψ characters, a byte string of size bytes is allocated. (This byte string will cover more than one word.)

In packed tables, it is possible to have a Hollerith or STC item begin at any byte of a word.

3.5.1.1.15.3 Representation of Items Within Bit and Byte String Containers

- Integer and Fixed Point Items:
  One binary digit is stored in one memory bit. The least significant binary digit of the magnitude is assigned to the rightmost memory bit, i.e., magnitudes are right-justified in their containing bit strings. A memory bit is set to 1 or 0 according as the binary (magnitude) digit is 1 or 0. The binary digit representing

the sign, if the Integer or Fixed-Point item is signed, is stored in the leftmost bit of the containing bit string. If the containing bit string contains a number of bits greater than the size of the item, the bits between the sign bit and the high order magnitude bits (called filler bits) are set to zero. Magnitudes of Integer and Fixed Point items are always represented as positive binary positional numbers regardless of the value of the sign.

Example:

In a 15 bit container

	Sign		Magnitude
+12 is represented as	0	0 0 0 0 0 0 0 0 0 0 0	1 1 0 0
-12 is represented as	1	0 0 0 0 0 0 0 0 0 0 0	1 1 0 0

Note that the representation of negative Fixed Point numbers in the Standard J3 memory differs from the 1's and 2's complement representation common to many computers.

The magnitude of an Integer or Fixed Point number must never exceed 2**(δ -1) -1.

- Dual Item:
  The bit string containing a Dual item is divided into two bit substrings equal in length. The first component of the item is assigned to the leftmost substring. Components are represented as Integer or Fixed Point data in their respective substrings.

  The magnitude of a component must not exceed 2** (δ/2 -1) -1.

- Floating Point Item:
  The characteristic and mantissa of a Floating-Point item are represented in the same manner as Integer and Fixed-Point items.

- Boolean Item:
  A Boolean item is represented as an unsigned integer with size of one binary digit in its containing bit string.

- Status Item:
  A Status item is represented as an unsigned integer within its containing bit string. Status items may never take more than 2** (δ -1) -1 values.

- Literal Items:
  If the size of the literal item is less than Ψ characters, the rightmost character in the character string is assigned to the rightmost byte in the containing string of Ψ bytes, i.e., character strings are right-justified in their containers. In this case, the leftmost Ψ - size of the literal item is greater than or equal to Ψ , character strings are left-justified in their byte string containers.

The encoding of Hollerith characters in terms of binary digit configurations within bytes is implementation dependent.

The following octal number / STC character configuration must obtain:

SIGN	CODE	SIGN	CODE
Space	0	X	35
A	6	Y	36
B	7	Z	37
C	10	)	40
D	11	-	41
E	12	+	42
F	13	=	44
G	14	$	47
H	15	*	50
I	16	(	51
J	17	,	56
K	20	0	60
L	21	1	61
M	22	2	62
N	26	3	63
O	24	4	64
P	25	5	65
Q	26	6	66
R	27	7	67
S	30	8	70
T	31	9	71
U	32	'	72
V	33	/	74
W	34	.	75

Q71 In your version of JOVIAL is the allocation of item data to memory performed exactly as described above?

Yes _____

No _____

If your implementation simulates the Standard J3 memory, please show the actual values in your implementation for

δ (number of bits/word)_____

Ψ (number of bytes/word)_____

ω (number of bits/byte)_____

Q71A Please describe any deviations below:

_____

_____

_____

_____

3.5.1.2 Structure Attribute = "Array"

Arrays are data aggregates. The most fundamental unit datum, or element, of an array is an item; item attributes (paragraph 3.5.1A) associated with each "element" in the array have identical values. Besides these item attributes, arrays have associated with them dimensionality and substructure size attributes.

Array data are defined in J3 by means of array declarations:

```
ARRAY     identifier                  (e.g., TWO'BY'TWO'MATRIX
          list'of'substructure'sizes          2  2
          item'description $                  F  $ )
          array'constant'list
```

(Throughout the remainder of this section, item'description is used to stand for
basic'item'description
[round'specification]
[range'specification])

3.5.1.2.1 Substructure Size Attribute

Substructure sizes are unsigned integer constants which serve to define the number of elements in an array substructure. The first integer in the list'of'substructure'sizes is the number of items in the most elementary substructure (e.g., number of items in a column vector; the second integer is the number of elementary substructures in the next higher substructure (e.g., number of columns in the matrix, etc.)

3.5.1.2.2 Dimensionality Attribute

The dimensionality attribute takes positive integers as values. The value, assigned by the array declaration, is the number of substructure size integers in the list'of' substructure'sizes. Standard J3 imposes no limit on dimensionality.

3.5.1.2.3 Allocation to Memory

If the Array is Integer, Fixed, Floating, Dual, Status, or Literal with size less than or equal to Ψ, each array element is allocated 1 word of memory as if the item element had been declared independently of the Array.

If the Array is Literal with size greater than Ψ, each element of the Array is allocated as many words as is required to cover the containing byte string.

If the Array is Boolean, the elements of each column (most elementary substructure) are stored up to δ per word, left to right, bit by bit, using as many computer words as required to contain them.

The order of storage of the elements is: first, by the elements of the column; second, the columns of a matrix; third, planes of a volume, etc.

In your version of JOVIAL, is the "Array" feature

Q72 included exactly as described above? _____

Q72A included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

Also, please state the reason or reasons why your implementation of the feature:

Q73 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q74 how many times are arrays declared in your program?

By count _____

By estimate _____

3.5.1.3 Structure Attribute = "Table"

Like arrays, tables are also data aggregates; broadly speaking, a table is a one dimensional array of table entries. Each entry may be composed of any number of items, and these items may be of differing types. The following attributes are associated with Tables:

Size
Variability
Basic Structure
Packing

Tables are declared by Table Declarations and Like Table Declarations.

3.5.1.3.1 The Table Declaration takes the form:

```
TABLE    [ identifier ]
         variability'specification
         size'specification
         [ basic'structure'specification ]
         [ table'packing'specification ]
         BEGIN entry'description END  $
```

An entry'description may consist of one or more occurrences of one or more of the forms:

```
         ITEM     identifier
                  item description
                  [ entry'packing'specification ] $
                  [ table constant'list ]

or       subordinate'overlay'declaration $
or       string'declaration $
         [string'constant'list ]
```

(For ordinary tables (i.e., tables with ordinary packing), only ITEM and overlay forms may appear in an entry'description; for defined tables, only ITEM and string forms may appear in an entry'description.) An example of a table declaration is

```
TABLE TAB'A V 10 S M $
  BEGIN
       ITEM IA'1  I 24 S $
       ITEM IA'2  F $
       ITEM IA'3 H 3 $
  END
```

In your version of JOVIAL, is the "Table" feature

Q75 included exactly as described above? _____

Q76 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard:

_____

_____

_____

_____

Q77 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q78 how many times are Tables used in your program?

By count _____

By estimate _____

3.5.1.3.2 The Like Table Declaration

has the form:

```
TABLE   identifier
        [size'specification ]
        [basic'structure specification ]
        [table'packing'specification ]
        L $
```

This declaration causes the compiler to generate a Table Declaration for the table named "identifier" by copying the Table Declaration of a "parent" table which has already been declared. The name of the parent table is identical to "identifier" of the like table, except that like-table "identifier" has one more digit or letter at the end. In the process of copying, the bracketed specifications in the Like Table Declaration, if included, override their correspondent specifications in the parent table. The entry'description of the parent table is copied intact, except that the letter or digit appended to the like table identifier is also appended to the item identifiers of the parent table when they are copied. An example of a Like Table Declaration is:

```
TABLE   TAB'AB L $
```

The parent table is TAB'A above; this Like Table Declaration has the effect of producing the Table Declaration:

```
TABLE  TAB'AB V 10 S M $
  BEGIN
       ITEM IA'1B  I 24 S $
       ITEM IA'2B  F $
       ITEM IA'3B  H 3 $
  END
```

(Note that a Like Table Declaration does not physically copy a Table Declaration into the source program listing.)

In your version of JOVIAL, is the "Like Table Declaration"

Q79 included exactly as described above? _____

Q80 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q81 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q82 how many times is the Like Table Declaration used in your programs?

By count _____

By estimate _____

3.5.1.3.3 Size Attribute

The size attribute is evaluated by the size'specification, an unsigned integer constant, and represents the number of entries in the table. When the table is allocated to core, the first word of the table contains the size value. This word is called the NENT of the table. Space is allocated for tables on the basis of the size value only; if the table is variable, size means the maximum number of entries in the table.

3.5.1.3.4 Variability Attribute

Variability = "variable" or "rigid" according as the variability'specification is V or R. The value of the NENT may be varied by the programmer if the table is variable; otherwise it may not.

In your version of JOVIAL, is the "Table Variability Attribute"

Q83 included exactly as described above? _____

Q84 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q85 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q86 how many times are tables declared rigid (R) in your programs?

By count _____

By estimate _____

3.5.1.3.5 Basic Structure Attribute

Basic structure = "serial" or "parallel" according as the optional'basic'structure' specification is S or P. If the basic'structure'specification is not included in the Table declaration, Basic structure = "parallel". If n denotes the size (number of entries) of the table, and m is the number of words required to represent all the items of a given entry, and if the table is serial, the table is allocated as n consecutive blocks of m words each. If the table is parallel, the table is allocated as m blocks of n words each.

In your version of JOVIAL, is the "Basic Structure" feature of Tables

Q87 included exactly as described above? _____

Q87A included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard:

Q88 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q89 how many times are tables declared "Parallel" in your programs?

By count _____

By estimate _____

Q89A how many times are tables declared "Serial" in your programs?

By count _____

By estimate _____

3.5.1.3.6 The Packing Attribute

Standard J3 provides the programmer with the facility to position - or pack - items in table entry words by allowing him to specify values for the Packing attribute. The packing attribute may be assigned two values - "ordinary" or "defined".

3.5.1.3.6.1 "Ordinary" Packing

"Ordinary" packing is declared when the table'packing'specification is either not included or appears as N, M, or D. N, M, or D represent respectively, "no packing", "medium packing", or "dense packing". If the table'packing'specification is not included, N is assumed. (In ordinary packing, entry'packing'specifications must not be included in the entry'description). "No", "Medium", and "Dense" packing have meanings which are implementation dependent, except that the following broad rules are to apply:

- "no packing" informs the compiler to allocate space for each item in an entry according to the same criteria used in allocating items declared independently of tables.

- "medium packing" directs the compiler to allocate more than one item to a word with the restrictions imposed that (non-literal) items do not overflow into the next word and that items do not share bytes.

- "dense packing" means the same as medium packing except that items may share bytes.

In your version of JOVIAL, is the "Ordinary Packing" feature of tables

Q90 included exactly as described above? _____

Q91 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q92 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

In how many table declarations in which ordinary packing is specified

Q93 do you have the packing'specification blank (implying "no packing")?

By count _____ By estimate _____

Q94 do you specify "no packing" by writing N

By count _____ By estimate _____

Q95 do you specify "medium packing" by writing M

By count _____ By estimate _____

Q95A do you specify "dense packing" by writing D

By count _____ By estimate _____

3.5.1.3.6.2 Defined Packing

The packing attribute is assigned the value "defined" when the table'packing'specification is an unsigned integer constant. This constant declares the number of words to be used per entry. Also, the entry'packing'specification in each table Item declaration must be entered. It is composed of two unsigned integer constants denoting respectively the word number in the entry in which the item is to be represented and the bit number (in that word) at which the item is to begin. It is also possible to enter N, M, or D following the two entry'packing'specification integers. In Standard J3, the item is allocated on the basis of the word and bit numbers only. When N, M, or D appear, they serve to describe the packing which results from the word and bit number integers and the situation of adjacent entries in the entry. "D" is assumed when none of the specifiers N, M, or D is entered.

An example of a defined Table Declaration is:

```
TABLE    TAB'A V 10 P 2 $
  BEGIN
      ITEM IA'1 I 24 S 0 0 $
      ITEM IA'1 F 1 0 $
      ITEM IA'3 H 3   0 24 $
  END
```

Note that Item IA'1 begins in word 0, bit 0
IA'2 word 1, bit 0
IA'3 word 1, bit 0

and that each entry is 2 words long.

In your version of JOVIAL, is the Defined Packing of tables feature

Q96 included exactly as described above? _____

Q97 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard:

_____

_____

_____

_____

Q98 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

Q99 how many times are Tables declared with Defined Packing in your programs?

By count _____

By estimate _____

3.5.1.4 Structure Attribute - "String"

A string is a substructure of an entry of a defined table (i.e., a table declared to have defined packing). It is declared by the string declaration within the entry's description of a Table Declaration:

```
STRING   identifier
         item'description
         word'no
         bit'no
          packing'specification
         occurrence'frequency
         beads'per'word $
          String'constant'list
```

The name of the string is "identifier". A string consists of one or more components called "beads". Word'no, bit'no, occurrence'frequency, and beads'per'word are unsigned integer constants. Word'no and bit'no specify the word number of the entry and the bit number in that word where the first bead begins. Also, bit'no tells the first bit of the first bead in each word where beads are allowed. If occurrence'frequency is n, then there are beads in every nth word of the entry. Beads'per'word specifies the number of beads in each word. The packing'specification is N, M, or D—these abbreviations have the same meaning as when used in ordinary tables.

An example of a String Declaration is:

```
TABLE  TAB'A  R  10  S  3  $
  BEGIN
    STRING SA 1  11  U
              0   1  D
              2   3  $
END
```

Note that a bead of a String is referenced by a two component subscript. SA (index'1, index'2 $) means bead number index'1 in entry number index'2.

In your version of JOVIAL, is the String Feature

Q100 included exactly as described above? _____

Q101 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the Standard?

_____

_____

_____

_____

Q102 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

Q103 how many times are strings declared in your program?

By count _____

By estimate _____

3.5.1.5 Structure Attribute = "File"

Files are declared by

```
FILE   identifier                    (e.g., INFILE
       file'structure'specification          H 10000 R 100
       status'constant'list                  V (BUSY) V (READY)
       device'name $                         DISKA)
```

Strictly speaking, File Declarations have the effect of declaring the datum named identifier as a status item whose values are those given in the status'constant'list. However, a file is also thought of as a set of "records" or groupments of data, and is therefore a data aggregate. The file'structure'specification has the form:

h'or'b recs'per'file r'or'v rec'length

H'or'b is either H, declaring a Hollerith file, or B, declaring a binary file. Recs'per' file is an unsigned integer constant representing the estimated maximum number of records in the file. R'or'v is either R or V according as all records in the file have the same size or records in the file have varying size. Rec'length is an unsigned integer constant specifying the maximum number of bytes (for Hollerith files) or bits (for binary files) in each record. Device'name is an implementation defined identifier specifying some input/output device. The status'constant'list reflects a set of conditions (e.g., "Busy", "ready", etc.) which may be exhibited by the device.

In your version of JOVIAL, is the File Feature

Q104 included exactly as described above? _____

Q105 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q106 not included at all? _____

If not included, kindly state the reason or reasons why.

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q107 how many times is the File Declaration used in your programs?

By count _____

By estimate _____

Q108 In how many instances where the File Declaration is used are files declared as Hollerith (h'or'b = H)?

By count _____

By estimate _____

Q109 In how many instances are Files declared to have variable length records (r'or'v = V)?

By count _____

By estimate _____

3.5.1.6 The Preset Attribute

Standard J3 permits the programmer to preset the values of data as they are declared to the compiler. Thus, for any datum, the value assigned to the preset attribute is just one of the possible values the datum itself can register. All data have some initial value when the program begins execution. If initial values are not specified by the programmer in data declaration, the initial value assigned is implementation dependent.

3.5.1.6.1

Item data are preset by including the item 'preset' specification in the Item Declaration. It has the form

P (+) constant (e.g., P + 3.14.59)

where + is used only if the item is numeric.

In your version of JOVIAL, is the Item Preset feature

Q110 included exactly as described above? _____

Q111 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

Also, please state the reason or reasons why your implementation differs from the standard?

Q112 not included at all? _____

If not included, kindly state the reason or reasons why.

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q113 how many times does presetting occur in Item Declarations in your programs?

By count _____

By estimate _____

3.5.1.6.2

Table items are preset by including the table'constant'list for one or more of the Item Declarations in a Table Declaration. The table'constant'list has the form:

BEGIN list'of'constants END (e.g., BEGIN 3H (ABC) 3H (XYZ) END)

where list'of'constants is one or more constants. The first constant of the list is assigned to the item in entry 0, the second to the item in entry 1, etc., until all the constants have been assigned. The number of constants in the list may be less than the number of entries in the table. Note that the table'constant'list follows directly the Item Declaration to which it applies.

In your version of JOVIAL, is the Table Presetting feature

Q114 included exactly as described above? _____

Q114A included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q115 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q116 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.1.6.3

Array items are preset by including the array'constant'list in the Array Declaration. If the array has dimensionality 1, the array'constant'list has the same form and effect as the table'constant'list. For arrays of dimensionality 2 or higher the array'constant' list has the form

BEGIN list'of'constants END
BEGIN list'of'constants END

The number of innermost BEGIN/END's must equal the first element in the list'of'sub-structure'sizes in the Array Declaration; the number of constants in each list'of'constants must not exceed the second element in the list'of'substructure sizes. For arrays of higher dimensionality, the same rules apply except that the 2 dimensional structure is replicated and BEGIN/END brackets inserted as required.

In your version of JOVIAL, is the Array Presetting feature

Q117 included exactly as described above? _____

Q118 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q119 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

Q120 How many times is Array Presetting used in your programs?

By count _____

By estimate _____

3.5.1.6.4

Strings are preset by supplying the string'constant'list in the String Declaration. The string'constant'list has precisely the same form as a two dimensional array' constant'list (described in the preceding paragraph).

In your version of JOVIAL, is the String Presetting feature

Q121 included exactly as described above? _____

Q122 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q123 not included at all? _____

If not included, kindly state the reason or reasons why.

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q124 How many times are strings preset in your programs?

By count _____

By estimate _____

3.5.1.7 Specification of Data Addresses and Data Sequencing

Ordinarily, word addresses for data in Standard J3 are determined automatically by the compiler. By means of OVERLAY statements, however, it is possible to specify absolute addresses and relative sequencing of data.

```
OVERLAY     constant'or'list'of'identifiers
                = list'of'identifiers     $
```

or

```
OVERLAY     list'of'identifiers     $
```

where list'of'identifiers is a sequence of one or more identifiers separated by commas.

Data are assigned sequential word addresses in the order in which their identifiers appear in the list'of'identifiers. Sequences separated by equal signs all begin at the same address. If constant'or'list'of'identifiers is a single octal of integer constant, that constant is the absolute address at which each sequence begins. The second form is used solely to direct sequencing.
In your version of JOVIAL, is the "Independent Overlay" feature

Q125 included exactly as described above? _____

Q126 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q127 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q128 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q129 How many times is the Independent Overlay used in your programs?

By count _____

By estimate _____

3.5.1.7.2

Subordinate Overlay Declarations apply only to tables with ordinary packing and must be inserted in the entry'description of the Table Declaration. Their form is identical to that of Independent Overlay Declarations except that only list'of'identifiers may be used (no absolute origin). The meaning is the same except that allocation applies to entries within a table (the identifiers must be for items declared in the entry description). An example of a subordinate overlay in a table declaration is

```
TABLE TAB'A V 10 S $
   BEGIN
       ITEM IA'I 1 23 S $
       ITEM I'A'2 F $
       ITEM IA'3
       ITEM IA'4 A 23 S 5 $
       OVERLAY IA'4 = IA'I $
END
```

In your version of JOVIAL, is the "Subordinate Overlay" feature

Q130 included exactly as described above? _____

Q131 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q132 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q133 How many times are Subordinate Overlays used in your programs?

By count _____

By estimate _____

3.5.2 Formulas

Formulas are essential building blocks in Standard J3. They may appear in IF Statements, as the right hand sides of assignment statements, in subscripts, as actual input parameters of functions and procedures. It is possible to distinguish eight kinds of formulas in J3: numeric, dual, Hollerith, Standard Transmission Code, entry, relational, and Boolean. Aside from arithmetic, relational, and logical operators, the most fundamental units of formulas are variables and formula constants.

3.5.2.1 Formula Constants

A formula constant is either a constant as described in Section 1 (e.g. 3.14, 3H (XYZ)) or one of the functional modifiers 'LOC, NENT, or NWDSEN acting on an identifier. Note that the value of the NENT of a rigid table is a constant (it's value cannot be changed during program execution) while the NENT of a variable table is a variable; for convenience sake, information regarding NENT is solicited under "Variables" below. The values of formula constants do not change during program execution - they are "used" but never "set".

3.5.2.1.1 'LOC

- form: 'LOC (data identifier) (e.g., 'LOC (TAB'A))
  or 'LOC (statement'label'identifier) (e.g., 'LOC (ADD'EM'UP.)
- meaning: 'LOC operating on an identifier produces an unsigned integer datum whose value is the machine address at which the named data or statement begins.

In your version of JOVIAL, is the " 'LOC" Functional Modifier

Q134 included exactly as described above? _____

Q134A included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

Q135 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q136 How many times is this feature used in your programs?

By count _____
By estimate _____

3.5.2.1.2 NWDSEN

- form: NWDSEN (table 'identifier) (e.g., NWDSEN (TAB'A))
  or NWDSEN (table 'item' identifier) (e.g., NWDSEN (II'A))
- meaning: The value produced represents as an unsigned integer the number of words per entry of the specified table.

In your version of JOVIAL, is the "NWDSEN" Functional Modifier

Q137 included exactly as described above? _____

Q138 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q139 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Q140 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.2.2 Variables

In J3, variables are:

- item identifiers, and subscripted array, string, and table identifiers,
- certain functional modifier expressions,

and

- function invocations.

Except for an ENTRY expression, each is an item datum. Values of variables may change throughout program execution; all but function invocations may appear on the left hand side of an assignment statement (e.g. VAR=3$), as well as on the right hand side.

3.5.2.2.1 Subscribing Variables:

Table, Table item, array, and string identifiers take numeric formulas of any complexity as subscripts. The number of numeric formulas which appear in a subscript must be one in the case of a Table Item or Entry Functional Modifier expression, two in the case of a String, and, in the case of an Array, the number of formulas must be equal to the dimensionality of the array. When the subscript is evaluated, it is converted, if necessary, to integer form before being used to fetch the element desired from its data structure. Subscript elements must never be less than 0 and should not exceed the limits of the Table, String, or Array. The compiler does not generate code to check for either contingency. In J3 it is possible to have nested subscripting to any depth, e.g., ALPHA ($II ($JJ ($KK + 1$) $) $). It should also be noted that subscripts may be formed with loop'variable formulas.

In your version of JOVIAL, is the Variable Subscripting feature

Q141 included exactly as described above? _____

Q142 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

Q143 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q144 In how many subscripts are complex numeric formulas used rather than single constants, variables or loop variables, e.g., subscripts like AA + BB * CC or AA + SIN (XX)/3.0?

By count _____

By estimate _____

Q145 In how many subscripts is nested subscripting used, e.g.,

AA ($ BB ($ II ($ K $) $) $)

By count _____

By estimate _____

3.5.2.2.2 Variable Functional Modifiers

3.5.2.2.2.1 BIT

Form: BIT ($ index $) (variable'name [subscript])
Meaning: variable'name must be subscripted if the variable'name is that of a table, string, or array. "Index" may have two components, each of which may be a full numeric formula. The first component is the number of the bit in the item referenced. The second component is the number of bits to be referenced, starting with the bit specified by the first component. If the second component is not present, 1 is assumed. The purpose of BIT is to define a bit substring within a named item. The substring is regarded as an unsigned integer when it appears in numeric formulas or assignment statements. BIT is undefined if the variable it modifies is Floating Point.

In your version of JOVIAL, is the BIT Functional Modifier

Q146 included exactly as described above? _____

Q147 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

Q148 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q149 How many times is BIT used in your programs?

By count _____

By estimate _____

3.5.2.2.2 BYTE

form: BYTE ($ index $)
(literal 'variable' name [subscript])

meaning: The meaning of BYTE is analogous to BIT, "byte" being substituted for "bit" in the definition. The effect of BYTE is to produce a substring of bytes. This substring is regarded as a Hollerith or STC datum according as the literal 'variable' name has been declared Hollerith or STC. Note that BYTE may be applied only to a Hollerith or STC datum.

In your version of JOVIAL, is the BYTE Functional Modifier

Q150 included exactly as described above? _____

Q151 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q152 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q153 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.2.2.2.3 CHAR

form: CHAR (floating 'variable' name [subscript])
meaning: CHAR extracts the characteristic portion of the Floating Point variable to which it is applied. The value obtained is treated as a signed integer datum.

In your version of JOVIAL, is the CHAR Functional Modifier

Q154 included exactly as described above? _____

Q155 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard:

_____

_____

_____

_____

Q156 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

Q157 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.2.2.2.4 MANT

form: MANT (floating 'variable' name [subscript])
meaning: MANT extracts the mantissa of the Specified Floating Point item. The value is treated as a signed fixed point datum all the bits which are fractional.

In your version of JOVIAL, is the MANT Functional Modifier

Q158 included exactly as described above? _____

Q159 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q160 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q161 How many times is MANT Functional Modifier used in your programs?

By count _____

By estimate _____

3.5.2.2.2.5 POS

form: POS (file ' name)
meaning: The value of POS is an unsigned integer ranging between 0 and K - 1, where K represents the number of records in the file called file name. POS (file'name) =k means that the next record which will be read or written (via INPUT, OUTPUT, OPEN, or SHUT) will be the kth record of the file. (k=0, 1,.....K-1). INPUT, OUTPUT, OPEN, or SHUT always add 1 to POS. If POS is set by an assignment statement, the statement POS (UFO'FILE) =0 causes the file to be positioned before the first record in UFO'FILE.

In your version of JOVIAL, is the POS Functional Modifier

Q162 included exactly as described above? _____

Q163 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementations differ from the standard?

Q164 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q165 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.2.2.2.6 NENT

form: NENT (table'name)
or NENT (table'item'name)
meaning: NENT is an unsigned integer whose value is the number of entries in the specified table. If the table is variable, NENT is a variable and can be set. If the table is rigid, NENT must be treated only as a constant.

In your version of JOVIAL, is the NENT Functional Modifier

Q166 included exactly as described above? _____

Q167 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q168 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the function of this feature?

_____

_____

_____

_____

Q169 How many times is NENT functional modifier programs?

By count _____

By estimate _____

3.5.2.3 Numeric Formulas

In J3 Numeric Formulas may be formed by connecting Fixed, Floating and Integer variables, subscripted as required, together with the arithmetic operators +, - , *,/ , ** (exponentiation). Numeric Formula Constants and single variables standing alone also constitute numeric formulas.
Examples of numeric formulas are:

```
 AA ($ 3 $)
-AA
 AA + BB * (CC - (DD**EE) )
(AA+BB)
```

3.5.2.3.1 Parenthesized Numeric Formulas

It is possible to bracket numeric formulas with parentheses to form numeric formulas. (CC - (DD**EE)) and (AA+BB) are parenthesized numeric formulas.

In your version of JOVIAL, is the Parenthesized Numeric Formula Feature

Q170 included exactly as described above? _____

Q171 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q172 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any which you employ to perform the functions of this feature?

Q173 How many times are parenthesized numeric formulas used in your programs?

By count _____

By estimate _____

3.5.2.3.2 Prefix + and -

In J3, two kinds of + and - operators are distinguished. A + or - is called infix if it is the first operator following an operand; otherwise it is a prefix.
Examples:

The + in AA + BB is infix.
The + in +AA is prefix.
The - in -(AA*CC) is prefix.
The - in AA-CC is infix.
The - in AA*-CC is prefix.

The prefix + has no effect on computations. The prefix - has the effect of multiplying the next operand by -1. Prefix + and - take precedence over all other operators. Thus -AA**BB is 9 if AA=3, BB=2.

In your version of JOVIAL, is the Prefix + and - feature

Q174 included exactly as described above? _____

Q175 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q176 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q177 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.2.3.3 Exponentiation

Exponentiation may be indicated either by ** or by brackets (* *), e.g., AA**BB may also be represented by AA (*BB*).

In your version of JOVIAL, is exponentiation notation

Q178 included exactly as described above? _____

Q179 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

Q180 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q181 How many times do you use (**) to indicate exponentiation?

By Count _____

By Estimate _____

Q181A How many times do you require the use of the exponentiation operator?

By Count _____

By Estimate _____

3.5.2.3.4 Absolute Values

Absolute values may be taken in J3 by using either ABS or the brackets (/ /), e.g.,

ABS (AA) or
(/ AA /)

In your version of JOVIAL, is the Absolute Value Feature

Q182 included exactly as described above? _____

Q183 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q184 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q185 How many times is (/ /) used to indicate absolute value in your programs?

By count _____

By estimate _____

3.5.2.3.5 Mixed Item Types

No restriction is imposed on mixing (numeric) item types in numeric formulas. For example, one might write

AA * BB + CC

where AA, BB, and CC are declared respectively as "Integer", "Fixed Point", and "Floating Point". Conversion to a like item type is performed automatically for the purposes of evaluating the formula.

In your version of JOVIAL, is the capability of mixing Item Types

Q186 included exactly as described above? _____

Q187 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

Q188 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q189 How many numeric formulas are there in your programs in which item types are mixed?

By count _____

By estimate _____

3.5.2.4 Dual Formulas

All the rules which apply to the formation and evaluation of numeric formulas apply as well to dual formulas, except that:

- Operations apply to each component of dual variables separately, and
- the appearance of numeric formulas within dual formulas is permitted; their resultant values are "twinned" before being evaluated in with dual formulas.

Examples:

If AA and BB are Dual and have respectively the values D(3,5), D(-1,1), then AA+BB results in the Dual number D(2,6). The formula AA+3 is equivalent to AA+D(3,3). (3 is "twinned").

In your version of JOVIAL, is the Dual Formula Feature

Q190 included exactly as described above? _____

Q191 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementationof the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q192 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q193 How many times do Dual Formulas appear in your programs?

By count _____

By estimate _____

3.5.2.5 Status, Hollerith and STC Formulas

All of these formulas have the form

{ Status / Hollerith / STC } 'Constant, or

{ Status / Hollerith / STC } 'Variable, or

{ Status / Hollerith / STC } 'Function.

Literal Formulas are either Hollerith or STC Formulas, or Octal Constants.

In your version of JOVIAL, are the Status, Hollerith, and STC Formulas

Q194 included exactly as described above? _____

Q195 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

Q196 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

3.5.2.6 Entry Formulas

Entry Formulas are either Entry variables or the constant 0. An entry variable is formed by operation of the functional modifier ENTRY on a subscripted table name or table item name:

ENTRY ({ table'name / table'item'name } ($ index $)).

The effect is to access entry number "index" of the specified table. The entry has no

item type associated with it; it is treated simply as a string of bits. An entry variable may appear on either side of the = sign in an assignment statement ENTRY (TAB'A ($ INDEX $)) = 0 has the effect of setting all bits in entry number INDEX of TAB'A to zero. Alternately, one may write ENT for ENTRY.

In your version of JOVIAL, is the Entry Variable and Formula Feature

Q197 included exactly as described above? _____

Q198 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q199 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q200 How many times is the ENTRY (or ENT) functional modifier used in your programs?

By count _____

By estimate _____

Q201 How many times is ENT used (instead of ENTRY) ?

By count _____

By estimate _____

3.5.2.7 Relational Formulas

Relational Formulas have the general form
formula relational' operator formula
or relational ' formula relational ' operator formula

where the relational'operator is one of EQ NQ LS LQ GR GQ. All formulas in a given relational formula must be of the same type, i.e., numeric, dual, status, literal, or entry. Examples of the first form are:

AA + 3 GQ SIN (XX)

ENTRY (A'TABLE ($ 0$)) EQ 0.

Examples of the second form are:

3.0 LS XX LS 4.0

SIN (XX) GR AA LS 1.0

Relational formulas are simple Boolean Formulas in the sense that if the two formulas are in fact in the indicated relation, the relational formula has the value "true"; if not, false. Relational formulas of the type shown in the second form are evaluated from left to right, two formulas at a time. Thus,

3.0 LS XX LS 4.0

is true if, and only if, 3.0 LS XX is true and XX LS 4.0 is true. The only operators permitted between Hollerith formulas are EQ and NQ.

In your version of JOVIAL, is the Relational Formula Feature (including chained formulas of the form 3.0 LS XX LS BB)

Q202 included exactly as described above? _____

Q203 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q204 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q205 How many times are "chained" relational formulas (e.g., 3.0 LS XX LS 4.0) used in your programs?

By count _____

By estimate _____

3.5.2.8 Boolean Formulas

A Boolean Formula may be a Boolean Constant (0 or 1), Boolean variable, Boolean function, Relation formula, Boolean Formulas connected together by AND or OR, or Boolean Formulas prefixed by NOT.
Boolean 'formula AND Boolean 'formula is true if and only if both Boolean 'formulas are true.
Booleans 'formula OR Booleans 'formula is true if one or both Boolean 'formulas are true.
NOT Boolean 'formula is true only if Boolean variables by enclosing them in parentheses, e.g., AA LS BB AND (CC GR 0 OR DD EQ 1).

In your version of JOVIAL, is the Boolean Formula Feature

Q206 included exactly as described above? _____

Q207 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q208 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q209 How many times are Boolean formulas containing AND, OR, or NOT used in your programs?

By count _____

By estimate _____

3.5.2.9 ODD

ODD is a functional modifier which designates a boolean variable.

ODD (numeric'operand)

produces the value "true" if the least significant bit of the numeric'operand is 1; otherwise, the value is "false". The numberic operand is either a loop'variable or a named'numeric'variable which is not floating.

In your version of JOVIAL, is the ODD feature

included exactly as described above? _____

included but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

Also, please state the reason or reasons why your implementation idffers from the standard?

_____

_____

not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

How many times do you use functional modifier ODD?

By Count _____

By Estimate _____

3.5.3 Data Transferral

In Standard J3, data are transferred from one part of main storage to another by means of the Assignment and Exchange Statements. Data are transferred between internal storage and external storage by means of the file manipulation statements, INPUT, OUTPUT, OPEN and SHUT.

3.5.3.1 The Assignment Statement

The Assignment Statement has the form:

named' { numeric / dual / literal / status / Boolean / Entry } 'variable =

{ numeric / dual / literal / status / Boolean / Entry } 'formula $

(where bracketed names must match, line by line). The effect is to transfer the value of the formula just evaluated to the named variable.

3.5.3.1.1

When numeric formulas are assigned to numeric variables, an automatic item type conversion takes place if the type of the formula is not the same as the type of the variable (e.g., "Integer" formula value is converted to "Floating Point" prior to assignment to "Floating Point" variable).

In your version of JOVIAL, is the Assignment Statement Item Type Conversion feature

Q210 included exactly as described above? _____

Q211 included but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

Also, please state the reasons why your implementation differs from the standard?

_____

_____

Q212 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

Q213 How many times are Assignment Statements which involve item type conversion used in your programs?

By Count _____

By Estimate _____

3.5.3.1.2

It is possible to have Boolean Assignment Statements in J3 of the type:

AA = BB LS 3.0 AND NOT CC

where AA and CC are Boolean Variables and BB is a numeric variable. AA is true (=1), in the example above, if BB is less than 3.0 and CC is false; if BB is greater than or equal to 3.0 and/or C is true, AA is false (=0).

In your version of JOVIAL, is the Boolean Assignment Feature, where the right hand side is a Boolean formula more complex than just a single Boolean variable or constant

Q214 included exactly as described above? _____

Q215 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q216 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

Q217 How many times do such Assignment Statements appear in your programs?

By count _____

By estimate _____

How many assignment statements of any kind appear in your programs?

By count _____

By estimate _____

3.5.3.2 Exchange Statements

Exchange Statements have the form:

named' { numeric / dual / literal / status / Boolean / Entry } 'variable = =

named { numeric / dual / literal / status / Boolean / Entry } 'variable $

effect is equivalent to three assignment statements, e.g.,

```
AA = = BB $
```

is equivalent to

```
TEMP = AA  $
AA = BB    $
BB = TEMP  $
```

where TEMP is some intermediate store.

In your version of JOVIAL, is the Exchange Statement Feature

Q218 included exactly as described above? _____

Q219 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q220 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

Q221 How many times are Exchange Statements used in your programs?

By count _____

By estimate _____

3.5.3.3 Input/Output

In Standard J3, all I/O centers around the file concept. (The File Declaration was discussed earlier.) The mechanisms provided to manipulate files are the OPEN, INPUT, OUTPUT, and SHUT statements. Records may read from or written into a file by each of these statements. It should be noted that reading and writing is only initiated by these statements - successful completion of these operations must be checked by the programmer. This is done by testing the (implementation specific) values of the status variable file'name.

3.5.3.3.1

OPEN { INPUT / OUTPUT } file'name { [input'operand] / [output'operand] } $

When INPUT or OUTPUT are used in this context, they declare the file to be "Input"-- records of the file may be read only -- or "Output" -- records of the file may be written only -- until the file is shut. OPEN initiates all file control activities and must be operated prior to any INPUT, OUTPUT or SHUT statements associated with the file. If input'operand or output'operand is included, an INPUT or OUTPUT statement (as described in the next two paragraphs) is automatically executed subsequent to all initialization processing.

3.5.3.3.2

INPUT file'name [input'operand] $

A read operation is initiated against the file file'name. The input'operand is the datum into which the input record is transferred. An input'operant is a variable, array name, table name, table entry (table'name ($ index $)), or range of table entries (written table'name ($ index ... index $)).

INPUT may be invoked only against a file previously declared as an input file in an OPEN statement.

3.5.3.3.3

OUTPUT file'name [output'operand] $

A write operation is initiated against the file file'name. The record is written from the output'operand. An output'operand means the same as input'operand or constant. An OUTPUT statement is only valid with respect to a file which when opened was declared as OUTPUT.

3.5.3.3.4

SHUT { INPUT / OUTPUT } file'name { [input'operand] / [output operand] } $

SHUT "deactivates" the file named file'name. As with OPEN, if either input'operand or output'operand is present a read or write is initiated.

3.5.3.3.5

Note that no conversion, editing, or rearrangement of data takes place when information is transmitted between the main memory and peripheral devices (except conversions of external to/from internal Hollerith encoding, where applicable.)

3.5.3.3.6

Note that the file is positioned at record k according as POS (file'name) =k, as previously discussed. The file may be repositioned at record j by POS (file'name) =j$. (If j=0, a "rewind" is initiated if it is meaningful for the associated device.) Each time a read or write occurs, 1 is added to POS (file'name) automatically.

In your version of JOVIAL, are the File Manipulating features (OPEN, SHUT, INPUT, OUTPUT, POS)

Q222 included exactly as described above? _____

Q223 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the features:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q224 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q225 How many times does the OPEN statement with the input or output operand appear in your programs?

By count _____

By estimate _____

Q226 How many times does the OPEN statement without the input or output operand appear in your programs?

By count _____

By estimate _____

Q227 How many times does the SHUT statement with the input or output operand appear in your programs?

By count _____

By estimate _____

Q228 How many times does the SHUT statement, without the input or output operand appear in your programs?

By count _____

By estimate _____

Q229 How many times does the INPUT statement appear in your program?

By count _____

By estimate _____

Q230 How many times does the OUTPUT statement appear in your program?

By count _____

By extimate _____

3.5.4 Program Sturctures

3.5.4.1 The Simple Statement

The simplest program structure in Standard J3 is the simple statement. Assignment, OPEN, SHUT, INPUT, and OUTPUT statements are examples of simple statements.

3.5.4.2 The Compound Statement

Compound Statements are sets of J3 statements grouped together by BEGIN and END brackets. A compound statement may contain simple statements or compound statements. An example of a compound statement is:

```
BEGIN
      AA = BB $
      CC = DD + 1 $
      IF XX = YY $
            BEGIN
                XX = CC $
                GOTO ALPHA $
            END
      GOTO BETA $
END
```

Note that this compound statement contains a compound statement (i.e., that set of simple statements between the second BEGIN and first END). No limit is imposed by Standard J3 on the degree of nesting of compound statements.

In your version of JOVIAL, is the Compound Statement feature

Q231 included exactly as described above? _____

Q232 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementations of the feature:

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

Q233 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

Q 234 How many times is this feature used in your programs?

By count _____

By extimate _____

3.5.4.3 Statement Labels

As data are given names for the purpose of making reference to them, so are statements. These statement names are cal led statement labels and have the form identifier.
Example: ADD'EM'UP. AA=BB+CC $

It is permissable to have more than one label for a given statement -- in fact no limit on the number of labels for a given statement is imposed in Standard J3.
Example: XX. ADD'EM'UP. YY. AA = BB + CC $.
Control may be passed to the assignment statement "AA = BB + CC" by referring to any one of its labels, as in

```
GOTO XX $
GOTO ADD'EM'UP $
GOTO YY $
```

In your version of JOVIAL, is the Statement Labelling feature

Q235 included exactly as described above? _____

Q236 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q237 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q238 How many times in your programs are there a multiplicity of labels for the same statement?

By count _____

By estimate _____

3.5.4.4 Direct Code

Standard J3 permits the insertion of blocks of direct code (machine language statements) in programs. Such blocks have the form:

DIRECT direct'code JOVIAL

Between DIRECT and JOVIAL brackets it is also allowable to have ASSIGN statements:

Form 1: ASSIGN A ([integer'constant]) = named'variable $

Form 2: ASSIGN named'variable = A ([integer'constant]) $

Form 1 causes the value of the machine accumulator to be transferred to the named' variable; Form 2 causes the value of the named'variable to be moved to the accumulator. The named'variable is declared in the program exterior to the direct code statements. The integer'constant if present represents a scale for the accumulator value (implying the item type of the accumulator is fixed point). If the integer' constant is not included, the accumulator is treated as floating point.

In your version of JOVIAL, is the Direct Code feature

Q239 included exactly as described above? _____

Q240 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q241 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q242 How many times is this feature used in your programs?

By count _____

By estimate _____

Q243 Can you describe briefly the principal functions performed by the direct code segments used in your programs.

In Standard J3 it is possible to declare closed subroutines within the body of a program. These closed subroutines are called Closes and Procedures. The set of statements comprising them are isolated from the other program statements; control cannot pass to them in the usual sequential way. Rather, closed subroutines are operated by invoking them explicitly from one or more points in the program.

3.5.4.5 The Close

The Close is defined by a Close Declaration:

```
CLOSE identifier $
  BEGIN
     statements
  END
```

Closes may reference data declared in the main program. If a Close is defined within a FOR-loop, it may reference loop'variables activated by the FOR.
A Close is invoked by

```
GOTO identifier $
```

where identifier is the Close name. When the Close terminates operation, control is returned to the next statement following the GOTO which invoked it, unless the Close itself directs control outside of its scope.

In your version of JOVIAL, is the Close feature

Q244 included exactly as described above? _____

Q245 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q246 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q247 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.4.6 The Procedure

A Procedure Declaration sets up a Closed program or subroutine much like a Close except that a Procedure may have input and/or output parameters. The Procedure Declaration has the form

procedure'heading
procedure'body

If no input or output parameters are to be associated with a procedure, the procedure' heading has the form

PROC identifier declaration 'list $

or

PROC identifier ( ) declaration'list $

where identifier is the namve of the procedure.

If PROC has only input parameters, the procedure'heading has the form:

PROC identifier (input'parameter'list) declaration'list $

If the PROC has only output parameters, the procedure'heading has the form:

PROC identifier (=output'parameter'list) declaration'list $

If the PROC has both input and output parameters, the procedure'heading has the form:

PROC identifier (input'parameter'list = output'parameter'list) declaration'list $

An input'parameter'list consists of one or more item names, table names, array names, or Close names (the Close name must be followed by a period), separated by commas. An output'parameter'list consists of one or more item names, table names, array names, or statement labels (followed by a period). Identifiers used as formal input parameters must not be used as formal output parameter identifiers. The declaration'list is one or more MOCE directives, DEFINE directives, data declarations, or Program Declarations. Except for Close names and statement labels, all input or output parameters must be declared by ITEM, TABLE, or ARRAY declarations either in the declaration'list or in the procedure'body prior to being referenced in procedural statements. Procedure'body takes the form:

BEGIN set'of'statements END

A Procedure Declaration must not be made inside a Procedure Declaration. Identifiers declaraed exterior to a Proc may be declared interior to the Proc with different meanings; within the Proc, interior declarations have precedence. For identifiers interior to the Proc which have no associated interior declarations, exterior declarations apply.

Procedures are invoked simply by writing the Proc name and supply actual input and/or output parameters as required. Actual paramenters must agree in number and type with the formal parameters as declared; no actual parameter may be left out. If a formal input parameter is

item name
table name
array name
close name , then the actual input parameter may be formula
table name
array name
close name .

Note that status constants are not permitted as formula. If a formal output parameter is

item name
table name
array name
statement name , then the actual output parameter may be variable
table name
array name
statement name .

Before a Proc is invoked, formulas are evaluated and assigned to their corresponding formal items. Actual table names, array names and close names, in effect, replace corresponding formal table, array, and close names. When the Proc terminates execution, values of formal output parameters are assigned to actual output parameters. If the Proc causes control to pass to a statement in the calling program which is not a statement in the output parameter list, the proper assignment of formal to actual output parameters does not take place.

In your version of JOVIAL, is the Procedure feature (both declaration and invocation)

Q248 included exactly as described above? _____

Q249 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q250 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

Q251 How many times are Procedures declared in your programs? (Where the procedures are not intended as functions)

By count _____

By estimate _____

Q252 How many Procedures are declared in your programs in which at least one input parameter is a Close name?

By count _____

By estimate _____

Q253 How many Procedures are declared in your programs in which at least one output parameter is a statement label (i.e., an alternate exit from the PROC?)

By count _____

By estimate _____

3.5.4.7 REMQUO

Standard J3 provides a built-in Proc which accepts two integers as input, divides one by the other, and yields as output the quotient and remainder of division. This Proc is named REMQUO and is defined as follows:

```
PROC    REMQUO    (NUM, DEN = QUO, REM) $
ITEM    NUM  I      δ    S    $
ITEM    DEN  I      δ    S    $
ITEM    QUO  I      δ    S    $
ITEM    REM  I      δ    S    $
BEGIN
  QUO = NUM/DEN
  REM  = NUM - QUO * DEN
END
```

(δ) is the maximum number of bits permitted for a signed integer number)

In your version of JOVIAL, is the built-in Proc REMQUO

Q254 included exactly as described above? _____

Q255 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q256 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any which you employ to perform the functions of this feature

_____

Q257 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.4.8 Function

A function is declared exactly as a Proc except that
- there may be no output parameter list, and
- an item declaration with the same name as the Proc is incorporated either in the declaration list or in the procedure body of the Proc.

Functions are invoked within formulas as in the example

```
XX ** 2 + SIN (YY)
```

The function call itself (SIN (YY) above) stands for the value output by the Proc on the basis of the input to that Proc.

In your version of JOVIAL, is the Function feature

Q258 included exactly as described above? _____

Q259 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q260 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you can employ to perform the functions of this feature?

Q261 How many times are functions declared in your programs?

By count _____

By extimate _____

3.5.4.9 REM

Standard J3 provides a built-in function which yields the remainder produced in the division of two integer numbers. The function is declared as

```
PROC   REM      (NUM, DEN)  $
ITEM   NUM      I 6  S      $
ITEM   DEN      I 6  S      $
ITEM   REM      I 6  S      $
BEGIN
       REM = NUM/DEN  $
       TEM = NUM - REM * DEN  $
END
```

(6) is the maximum number of bits which may be used in an integer number)

In your version of JOVIAL, is the REM built-in function

Q262 included exactly as described above? _____

Q263 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q264 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any which you employ to perform the functions of this feature?

_____

Q265 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.4.10 Program

A program is declared in one of two ways.

Form 1:
```
START [origin]  $
list'of'statements
TERM  [statement'name] $
```

Form 2:
```
CLOSE identifier  $
START [origin] $
list'of'statements
TERM $
```

Form 1 declares an "independent" program (with no name associated). Origin is an integer or octal constant declaring the first word address of the program; if not included, the compiler determines the first word address. If statement name is included, execution begins at the statement labelled statement'name when the monitor invokes the program. Otherwise execution begins at the first executable statement. Form 2 declares a closed subroutine which may be invoked by other programs by a GOTO statement, i.e., Form 2 declares a "global" CLOSE. Origin has the same meaning as in Form 1.

In your version of JOVIAL, is Form 1 of this feature

Q266 included exactly as described above? _____

Q267 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

Also, please state the reason or reasons why your implementation differs from the standard?

Q268 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any which you employ to perform the functions of this feature?

In your version of JOVIAL, is Form 2 of this feature

Q269 included exactly as described above? _____

Q270 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q271 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any which you can employ to perform the functions of this feature?

3.5.4.11 Referencing External Programs

Communication between the present programs and one which is defined externally is established by means of the External Program Declaration:

```
PROGRAM identifier  [origin] $
```

The origin is either an unsigned integer or an octal number. The purpose of the "identifier" in Program Declaration is to inform the compiler the "identifier" is the name of closed subroutine compiled independently. This subroutine may have been compiled under Form 2 of the Program Declaration as described in the preceding section. Example:

If Program BB is declared by

```
CLOSE BB $
  START $
    statement of program'BB
TERM  $
```

then in order to invoke BB from Program AA, the External Program Declaration

```
'PROGRAM BB $
```

must be made in the calling program.

Origin is either an unsigned integer constant or an octal constant which represents the first word address of the declared program. Whether or not origin is included is a system dependent matter. If your system does not include the origin capability, it is not regarded as deviating from the Standard.

External programs are invoked by the statement

```
GOTO identifier $
```

The assumption is that the external program named "identifier" returns control to your program at the statement immediately following the invocation. The standard does not permit input and output parameters to be associated with the external program.

Note that an external program declared by an External Program Declaration need not have been defined as a "global" close as was Program BB in the example. However, any such program must

- return control to the calling program at the statement immediately following the GOTO invocation, and
- must not disturb loop variables active in the calling program at the time of invocation.

In your version of JOVIAL, is the Program Declaration feature

Q272 included exactly as described above? _____

Q273 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

Q274 not inluded at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

Q275 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.5 Sequence Control

Statements in Standard J3 are normally executed in the order in which they are written. Sequencing can be altered in a variety of ways, however. The simplest way is by means of the statement

```
GOTO statement'label  $
```

Q276 How many times is the simple "GOTO statement'label" statement used in your programs?

By count _____

By estimate _____

The GOTO statement may also be employed using indexed and item switches.

3.5.5.1 The Item Switch

The Item Switch declaration has the form:

```
SWITCH switch'identifier (name) =
        (item'switch'list)     $
```

Switch'identifier is the name of the switch.
Name is either an item identifier or a file identifier.
Item'switch'list is one or more occurrences of . . .

```
constant = sequence'designator
```

each occurrence set off from the others by commas.
No limit is imposed on the number of sequence'designators. A sequence designator takes the form

```
identifier
```

or,

```
identifier ($ index $)
```

where identifier is a statement label, program name, close name, or switch name. (When identifier ($ index $) is used, identifier must be a switch name.) The constants of the item'switch'list are possible values which may be taken by the item or file identifier name. If name is a file identifier, the constants must be the status constants declared in the File Declaration. An example of an Item Switch declaration is:

```
SWITCH ALPHA  (BETA)  =
        (3H (ABC)       = X1, 3H(DEF) = X2,
                3H(GHI) = X3)                $
```

(ALPHA is the switch name, BETA is an item name, X1, X2, X3 are statement, program, close or switch names, and the three Hollerith constants are possible values which BETA may take.)

Note that an Item Switch declaration may, like a close, be placed anywhere in the program - the computer effects a transfer around it.

Item switches are invoked by

GOTO switch'identifier [($ index $)] $.

($ index $) is required if the item is a subscripted variable. In that case, ($ index $) is the subscript for the item. The effect of an item switch invocation is to cause the current value of the item or file name to be compared against the constants in the item'switch' list. If a match is made, control is passed to the associated sequence designator. If no match is made, the next statement following the invocation is executed.

Example: GOTO ALPHA (3) causes control to pass to X2 if BETA (3) = 3H(DEF) —BETA is assumed to be a table item or a one dimensional array and (3) applies to BETA.

In your version of JOVIAL, is the Item Switch feature

Q277 included exactly as described above? _____

Q278 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q279 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q280 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.5.2 The Index Switch

The Index Switch is declared by SWITCH switch'identifier = (index'switch'list) $. Index'switch'list is one or more occurrences of sequence'designators separated by commas. No limit is imposed on the number of sequence'designators in the index' switch'list. Also it is possible to leave sequence designators out of the list, as in the example:

SWITCH GAMMA = (,X1, X2, , X3) $

Sequence'designators have the same form as for item switches. They may be statement, switch, or close names but not program names. Like an item switch, an index switch declaration may be included at any point in the program. The compiler effects a jump around it.

An index switch is invoked by . . .

GOTO switch'identifier ($ one'component'index $) $

The effect of the invocation is to pass control to the sequence'designator at position k in the index'switch'list if the value of the one'component'index is k. If k is not within the range of the index'switch'list, or if position k does not contain a sequence' designator, control is passed to the next statement following the invocation.

In your version of JOVIAL, is the index switch feature

Q281 included exactly as described above? _____

Q282 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q283 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the function of this feature?

_____

Q284 How many times are index switches used in your programs?

By count _____

By estimate _____

For both index and item switches,

Q285 in how many switch declarations are other switch names used as sequence'designators?

By count _____

By estimate _____

Q286 in how many are close names used?

By count _____

By estimate _____

Q287 in how many are program names used?

By count _____

By estimate _____

3.5.5.3 The IF Statement

The IF Statement has the form:

IF Boolean'formula $

Example:

IF AA+3 EQ BB AND CC LS 1.0 $

If, when evaluated, the Boolean'formula is true, control is passed to the first simple or compound statement following the IF statement. If the Boolean'formula is false, control is passed to the second simple or compound statement following the IF statement.

In your version of JOVIAL, is the IF statement

Q288 included exactly as described above? _____

Q289 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q290 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q291 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.5.4 IFEITH and ORIF

IFEITH statements have the form:

```
IFEITH  Boolean'formula  $
```

ORIF statements have the form:

```
ORIF Boolean statements  $
```

IFEITH and ORIF statements are linked together in "chains" of the form:

```
IFEITH'statement  $
  simple'or'compound statement  $
ORIF'statement
  simple'or'compound statement  $
  .
  .
  .
ORIF's statement
  simple'or'compound statement  $
```

As in IF statements, the Boolean'formulas appearing with IFEITH and ORIF are evaluated. If true, control is passed to the simple'or'compound'statement following, and when that statement terminates execution, control is passed to the statement immediately following the last such simple'or'compound'statement in the chain, provided, of course, that the simple'or'compound'statement doesn't itself transfer control elsewhere. If false, control is passed to the next ORIF or the statement immediately following the last simple'or' compound'statement in the chain. An example of an IFEITH/ORIF chain is:

```
        IFEITH   AA GR  BB  $
                 AA = BB + 1  $
L1.     ORIF     AA EQ  BB  $
        L2.      CC = DD  $
        ORIF     AA LS BB  $
                 GOTO ZETA  $
```

Note that statements inside the chain may be labelled; if control is transferred to a labelled ORIF (e.g., L1) from outside the chain the effect is the same as if control "dropped through" from the initial IFEITH. If control is passed to a labelled simple' or'compound'statement (e.g., L2), the effect is the same as if control "dropped through" to the preceeding ORIF and that statement was true. No limit is imposed on the number of ORIF's in the chain.

In your version of JOVIAL, is the IFEITH/ORIF

Q292 included exactly as described above? _____

Q293 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q294 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q295 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.5.5 Iteration Control

In standard J3, iterations may be controlled either by IF statements and counter incrementing statements or by means of FOR loops. There are three kinds of FOR statements:

3.5.5.5.1

The One-factor FOR statement has the form:

```
FOR loop'variable = numeric'formula $.
```

The effect is to activate the loop variable and to assign as an initial value the value of the numeric'formula. (A loop'variable is written as a single letter.)

3.5.5.5.2

The Two-factor FOR statement has the form:

```
FOR loop'variable = numeric'formula'1,
                    numeric'formula'2 $
```

The effect is incrementation value is declared having the value of numeric'formula'2.

3.5.5.5.3

The Complete FOR statement has the form:

```
FOR loop'variable = numeric'formula'1,
                    numeric'formula'2,
                    numeric'formula'3 $
```

The effect is the same as the two-factor FOR statement; additionally, an iteration limit is declared as having the value of numeric'formula'3. (As alternate form of the complete loop statement is discussed below with functional modifier ALL.)

3.5.5.5.4

The scope of a FOR statement begins at the FOR statement itself and ends at the end of the next non-FOR statement, either simple or compound, in sequence. Note that several FOR statements may be grouped together. Loop variables are active in the scope of the FOR statement declaring them. When execution terminates in the FOR scope,

- in the case of the one-factor FOR, control passes to the next statement;
- in the case of the two-factor FOR, control, the loop'variable is incremented by numeric'formula'2 and control passes to the next statement;
- in the case of the complete FOR, the loop'variable is incremented by numeric'formula'2 and compared to numeric'formula'3. If the loop'variable is "beyond" numeric'formula'3, control passes to the next statement. Otherwise, control is passed to the first statement after the FOR. "Beyond" means "greater" or "less", according as the increment is currently positive or negative

Loop variables are deactivated only when control is transferred outside the loop statement by a GOTO, as a statement label or close, or by normal loop termination.

Calling a Proc, Function or Program within the loop does not deactivate loop variables so long as the Proc, Function, or Program returns control.

In your version of JOVIAL, is the FOR loop feature

Q296 included exactly as described above? _____

Q297 included, but differing in one or more particulars from the standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q298 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q299 How many times are one-factor FOR loops used in your programs?

By count _____

By estimate _____

Q300 How many times are two-factor FOR loops used in your programs?

By count _____

By estimate _____

Q301 How many times are complete FOR loops used in your programs?

By count _____

By estimate _____

Q302 How many times do you use FOR loops to increment backwards? (as in FOR 1 = 10, -1, 0 $)

By count _____

By estimate _____

3.5.5.5.5 Functional Modifier ALL

It is possible to write a complete FOR statement in the form

```
FOR loop'variable = ALL (name) $
```

where name is either a table identifier or a table item identifier. This statement is equivalent to

```
FOR loop'variable = NENT (name) -1, -1, 0 $
```

In your version of JOVIAL, is the ALL function modifier

Q303 included exactly as described above? _____

Q304 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

Q305 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q306 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.5.5.6

The TEST statement has two forms:

```
TEST $       and
TEST loop'variable $.
```

TEST may appear only inside FOR loops. Its function is to transfer control within the loop to the increment/test control code which appears next (Form 1) or to the control code for the loop'variable.

In your version of JOVIAL, is the

Q307 included exactly as described above? _____

Q308 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q309 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q310 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.5.6

The RETURN Statement has the form:

RETURN $

and is meaningful only in CLOSES, Procs, and Functions. RETURN causes control to be transferred from these closed subroutines back to the next statement following the subroutine invocation. A RETURN statement is not required at the physical end of a subroutine declaration, since one is assumed to be there implicitly.

In your version of JOVIAL, is the RETURN feature

Q311 included exactly as described above? _____

Q312 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

Also, please state the reason or reasons why your implementation differs from the standard?

Q313 not included at all? _____

If not included, kindly state the reason or reasons why:

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

Q314 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.5.7

The STOP Statement has two forms:

```
STOP  $
STOP  statement'label  $
```

If STOP $ appears in an independently compiled closed subroutine (see Form 2, 3.5.3 K) it causes control to be returned to the calling program. If STOP $ appears in an independent (main) program (see Form 1, 3.5.3 K), control is returned to the monitor if there is one. If there is no monitor, the computer halts. On restarting, execution begins at the statement following STOP. If STOP statement'label appears anywhere in a J3 program structure, the computer is halted. When restarted, execution begins at the statement named statement'label.

In your version of JOVIAL, is the STOP statement feature

Q315 included exactly as described above? _____

Q316 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

_____

_____

Q317 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

_____

_____

Q318 How many times is the "STOP statement'label" form used in your programs?

By count _____

By estimate _____

Q319 How many times is the "STOP" form used in your programs?

By count _____

By estimate _____

3.5.6 Compool and the Define Directive

3.5.6.1 Compool

The notion of Compool (Communications Pool) in Standard J3 is that it is a dictionary containing definitions and locations of data, files, programs, etc. Compool does not contain values of data or complete binary programs — this information is maintained in in a library or data base. All identifiers appearing in Standard J3 programs need not be declared if definitions exist for them in the Compool; however, Compool definitions may be overridden by declarations for identifiers in the program.

Q320 Would you please state whether or not your JOVIAL system has some kind of Compool facility. _____

Q321 If so, please describe the characteristics of your Compool and the relationship between it and the rest of the system.

_____

_____

_____

_____

Q322 Do definitions in your programs override Compool definitions? _____

3.5.6.2 The Define Directive

has the form:
DEFINE identifier "string'of'J3'symbols" $
where string'of'J3' symbols means a string of identifiers, constants, and delimiters (i.e., non-alphabetic or numeric characters, primitives such as NENT, etc.) The purpose of DEFINE is to link the identifier with the symbol string so that each time the identifier is scanned in compilation, it is replaced by the symbol string. Example:

```
DEFINE  AA "BB * SIN (EE**XX)" $
```

The statement

```
YY = AA  $
```

is read by the compiler as

```
YY = BB * SIN (EE**XX)  $
```

DEFINE's may appear anywhere in the program; more than one DEFINE may appear for the same identifier (the second cancels the first, etc.). DEFINED identifiers may appear in the symbol string of a DEFINE, i.e., nesting to any depth is allowed. However, circular nesting is not permitted.

In your version of JOVIAL, is the Define Directive feature

Q323 included exactly as described above? _____

Q324 included, but differing in one or more particulars from this standard described above? _____

If yes, please describe your implementation of the feature:

_____

_____

Also, please state the reason or reasons why your implementation differs from the standard?

_____

_____

Q325 not included at all? _____

If not included, kindly state the reason or reasons why:

_____

_____

Also, can you describe other mechanisms, if any, which you employ to perform the functions of this feature?

_____

_____

Q326 How many times is this feature used in your programs?

By count _____

By estimate _____

3.5.7 Other Features

Please list here other features, if any, in your version of JOVIAL which are not specified in the standard. Do not include subroutines; rather, describe actual JOVIAL statements such as

"MERGE file'name with file'name ON key'name"

for example.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Data Dynamics, Inc. 9800 S. Sepulveda Boulevard Los Angeles, California 90045	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP N/A

3. REPORT TITLE

JOVIAL APPLICATION QUESTIONNAIRE

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Final Report

5. AUTHOR(S) *(First name, middle initial, last name)*

William M. O'Brien

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1968	158	2

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
F19628-68-C-0301 b. PROJECT NO. c. d.	ESD-TR-68-454 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Command Systems Division, Electronic Systems Division, Air Force Systems Command, USAF, L G Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

The JOVIAL Application Questionnaire was produced as a vehical to gather information regarding JOVIAL users experience with the language and the environment in which JOVIAL was being used. This information is to be utilized to evaluate JOVIAL (J3) computer programming languages as specified in AFM 100-24. The questionnaire contains: Instruction on how to fill out the questionnaire; general questions about the application being programmed in JOVIAL; the hardware and operating systems being used; background information; specific questions about each JOVIAL feature with regard to the conformance of the specification of the feature to AFM 100-24 and the extent of utilization of the feature. In addition, the questionnaire contains a detailed description of each JOVIAL feature as specified in AFM 100-24 as a convenient reference to the users of a different JOVIAL language dialect.

DD FORM 1 NOV 65 1473

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Programming Language Evaluation JOVIAL Evaluation						